Algorithmique Avancée

TD 3-4 : Arbres bicolores et tries

Table des matières

1 Arbres bicolores

Exercice 1.1 ***: Arbres 2-3-4***

Question 1.1.1 Rappeler les règles d'éclatement d'un 4-nœud dans un arbre 2-3-4, lorsque les éclatements se font à la descente.

Question 1.1.2 Construire par adjonctions successives un arbre 2-3-4 contenant les clefs 8, 3, 2, 4, 1, 15, 10, 9, 11, 7, 6, 13, 12, 5, 14, 16, 17. On nommera cet arbre `A-ex1`.

Définition des arbres bicolores

Un *arbre bicolore de recherche* est un arbre binaire de recherche complété dans lequel tout sommet possède une couleur (*blanc* ou *rouge*) et tout nœud interne possède une clef et qui vérifie :

(a) la racine est blanche
(b) les feuilles sont blanches (et ne possèdent pas de clef)
(c) le père d'un sommet rouge est blanc
(d) les chemins issus d'un même sommet et se terminant en une feuille ont le même nombre de sommets blancs.

Primitives

Pour manipuler les arbres bicolores, on a des primitives habituelles sur les arbres binaires :

`ArbreVide`	*Rien* $\rightarrow$ *arbre binaire*
`ArbreVide()`	*renvoie un arbre vide*
`ArbreBinaire`	*élément* $\times$ *arbre binaire* $\times$ *arbre binaire* $\rightarrow$ *arbre binaire*
`ArbreBinaire(x,G,D)`	*renvoie l'arbre binaire dont la racine a pour contenu l'élément x et dont les sous-arbres gauche et droit sont respectivement G et D*

`EstVide`	*arbre binaire* → *booléen*
`EstVide(T)`	*renvoie VRAI ssi T est un arbre binaire vide*
`Racine`	*arbre binaire* → *élément*
`Racine(T)`	*renvoie le contenu de la racine de l'arbre binaire T*
`SousArbreGauche`	*arbre binaire* → *arbre binaire*
`SousArbreGauche(T)`	*renvoie une copie du sous-arbre gauche de l'arbre binaire T*
`SousArbreDroit`	*arbre binaire* → *arbre binaire*
`SousArbreDroit(T)`	*renvoie une copie du sous-arbre droit de l'arbre binaire T*

et des primitives spécifiques aux arbres bicolores :

`Arbre`	*clef* × *couleur* × *arbre binaire* × *arbre binaire* → *arbre binaire*
`Arbre(a,c,G,D)`	*renvoie l'arbre binaire dont la racine a pour clef a et pour couleur c et dont les sous-arbres gauche et droit sont respectivement G et D*
`Clef`	*arbre binaire* → *clef*
`Clef(T)`	*renvoie la clef de la racine de T*
`Couleur`	*arbre binaire* → *couleur*
`Couleur(T)`	*renvoie la couleur de la racine de T*
`ModifierCouleur`	*arbre binaire* × *couleur* → *Rien*
`ModifierCouleur(T,c)`	*remplace par c la couleur de la racine de l'arbre binaire T*
`FeuilleBlanche`	*Rien* → *arbre binaire*
`FeuilleBlanche()`	*renvoie l'arbre binaire réduit à une feuille de couleur blanche sans clef*
`EstFeuilleBlanche`	*arbre binaire* → *booléen*
`EstFeuilleBlanche(T)`	*renvoie VRAI ssi T est un arbre bicolore réduit à une feuille blanche sans clef*

Exercice 1.2 ***: Transformation d'un arbre 2-3-4 en arbre bicolore***

Principe : Pour obtenir un arbre bicolore à partir d'un arbre 2-3-4, on procède de la façon suivante :

- l'arbre vide est transformé en une feuille blanche (sans clef)
- chaque 2-nœud prend la couleur blanche et ses deux sous-arbres sont transformés en arbres bicolores
- un 3-nœud, qui compte deux clefs $a < b$, se scinde en deux nœuds de l'arbre bicolore ; ces deux nœuds contiennent respectivement les clefs a et b. Il y a deux possibilités : ou bien b est à la racine du sous-arbre droit de a ou bien a est à la racine du sous-arbre gauche de b. Le fils prend la couleur rouge et le père prend la couleur blanche. Les trois sous-arbres du 3-nœud sont eux-mêmes transformés en arbres bicolores et deviennent les sous-arbres des nœuds contenant les clefs a et b
- un 4-nœud, qui compte trois clefs $a < b < c$, se scinde en trois nœuds de l'arbre bicolore ; ces trois nœuds contiennent respectivement les clefs a, b et c. La clef a est à la racine du sous-arbre gauche de b et la clef c est à la racine du sous-arbre droit de b. Les fils prennent la couleur rouge et le père prend la couleur blanche. Les quatre sous-arbres du 4-nœud sont eux-mêmes transformés en arbres bicolores et deviennent les sous-arbres des nœuds contenant les clefs a et c.

Question 1.2.1 Illustrer le principe énoncé ci-dessus (au moyen de petits dessins).

Question 1.2.2 Construire un arbre bicolore, que l'on nommera `B-ex1`, transformé de l'arbre 2-3-4 `A-ex1`.

Question 1.2.3 Écrire l'algorithme de transformation d'un arbre 2-3-4 en arbre bicolore. On dispose des primitives des arbres 2-3-4 du cours.

Question 1.2.4 Montrer que l'arbre ainsi obtenu est un arbre bicolore et encadrer la hauteur de cet arbre.

Exercice 1.3 ***: Transformation d'un arbre bicolore en arbre 2-3-4***

Question 1.3.1 Donner le principe de la transformation d'un arbre bicolore en arbre 2-3-4. Illustrer ce principe au moyen d'un exemple.

Question 1.3.2 Écrire l'algorithme de transformation d'un arbre bicolore en arbre 2-3-4. La création d'arbres 2-3-4 sera assurée par des primitives `ArbreVide234`, et `Arbre234` qui prend en argument une liste de clés et une liste d'arbres 2-3-4.

Exercice 1.4 ***: Rotations dans un arbre binaire***

On définit les rotations simples `RotationGauche` et `RotationDroite` de spécifications :

`RotationGauche`	*arbre binaire → arbre binaire*
`RotationGauche(T)`	*renvoie l'arbre obtenu en faisant basculer T vers la gauche*
`RotationDroite`	*arbre binaire → arbre binaire*
`RotationDroite(T)`	*renvoie l'arbre obtenu en faisant basculer T vers la droite*

ainsi que les rotations doubles `RotationGaucheDroite` et `RotationDroiteGauche` de spécifications :

`RotationGaucheDroite :`	*arbre binaire → arbre binaire*
`RotationGaucheDroite(T)`	*renvoie l'arbre obtenu en faisant basculer d'abord le sous-arbre gauche de T vers la gauche puis l'arbre T ainsi modifié vers la droite*
`RotationDroiteGauche :`	*arbre binaire → arbre binaire*
`RotationDroiteGauche(T)`	*renvoie l'arbre obtenu en faisant basculer d'abord le sous-arbre droit de T vers la droite puis l'arbre T ainsi modifié vers la gauche*

Question 1.4.1 Écrire la définition de l'une des deux rotations simples et la définition de l'une des deux rotations doubles.

Exercice 1.5 ***: Insertion dans un arbre bicolore***

Principe : L'insertion dans un arbre bicolore suit le principe de l'insertion dans un arbre 2-3-4. Nous travaillerons ici sur l'insertion avec éclatements à la descente.

Question 1.5.1 Transposer dans les arbres bicolores les différents cas d'insertion d'une clef dans une feuille d'un arbre 2-3-4.

Question 1.5.2 Transposer dans les arbres bicolores les différents cas d'éclatement d'un 4-nœud.

Question 1.5.3 Réaliser l'insertion des clefs 13.1, 13.3, 12.5, 12.8 dans l'arbre bicolore `B-ex1`.

Question 1.5.4 Écrire l'algorithme d'insertion d'une clef dans un arbre bicolore.

Question 1.5.5 Ci-dessous figure un algorithme d'insertion extrait du livre "Introduction à l'algorithmique" de Cormen, Leiserson, Rivest et Stein. Est-ce le même que le nôtre ?

Algorithmes d'insertion dans un ABR et dans un arbre bicolore

```
ARBRE-INSERER(T,z)
  y <- NIL
  x <- racine[T]
  tantque x <> NIL
    faire y <- x
          si cle[z] < cle[x]
             alors x <- gauche[x]
             sinon x <- droit[x]
  p[z] <- y
  si y = NIL
     alors racine[T] <- z
     sinon si cle[z] < cle[y]
              alors gauche[y] <- z
              sinon droit[y] <- z
```

```
ROTATION-GAUCHE(T,x)
  y <- droit[x]
  droit[x] <- gauche[y]
  si gauche[y] <> NIL
     alors p[gauche[y]] <- x
  p[y] <- p[x]
  si p[x] = NIL
     alors racine[T] <- y
     sinon si x = gauche[p[x]]
              alors gauche[p[x]] <- y
              sinon droit[p[x]] <- y
  gauche[y] <- x
  p[x] <- y
```

Le code de ROTATION-DROITE est similaire au code de ROTATION-GAUCHE.

```
RN-INSERER(T,x)
  ARBRE-INSERER(T,x)
  couleur[x] <- ROUGE
  tantque x <> racine[T] et couleur[p[x]] = ROUGE
    faire si p[x] = gauche[p[p[x]]]
              alors y <- droit[p[p[x]]]
                    si couleur[y] = ROUGE
                       alors couleur[p[x]] <- NOIR
                             couleur[y] <- NOIR
                             couleur[p[p[x]]] <- ROUGE
                             x <- [p[x]]
                       sinon si x = droit[p[x]]
                                alors x <- p[x]
                                      ROTATION-GAUCHE(T,x)
                             couleur[p[x]] <- NOIR
                             couleur[p[p[x]]] <- ROUGE
                             ROTATION-DROITE(T,p[p[x]])
                       sinon (comme la clause alors
                                en echangeant droit et gauche)
  couleur[racine[T]] <- NOIR
```

2 Arbres de Recherche versus Tries

Exercice 2.1 ***: Comparaisons sur des exemples***

Question 2.1.1 Construire l'arbre binaire de recherche (ABR) obtenu par l'insertion successive des lettres : A, S, E, R, C, H, I, N, G, X, M, P et L.
On utilise l'ordre alphabétique pour comparer deux lettres.

Question 2.1.2 Que se passe-t-il si on construit l'ABR avec la succession suivante de lettres : L, S, A, R, E, H, C, N, I, X, G, P et M ?

Question 2.1.3 En utilisant le codage ci-dessous, et les deux exemples précédents, parmi les modèles d'ABR, d'arbre digital et d'arbre lexicographique lesquels sont sensibles à l'ordre d'insertion des lettres ?

A	00001
S	10011
E	00101
R	10010
C	00011
H	01000
I	01001

N	01110
G	00111
X	11000
M	01101
P	10000
L	01100

Remarque : on lit le bit de poids fort (le plus à gauche) d'abord.

Question 2.1.4 Construire les arbres binaire de recherche, digital et lexicographique avec la succession de lettres : N, A, S, P, I, X.

Exercice 2.2 ***: R-trie ou arbres de la Briandais ?***

On se place sur l'alphabet à 4 lettres A,C,G et T encodant les séquences d'ADN. On rappelle qu'un nœud d'un R-trie contient une valeur non vide lorsqu'il représente une clé. Cela permet notamment d'encoder deux mots dont l'un est préfixe de l'autre. On représente le 4-trie vide par $\emptyset$.

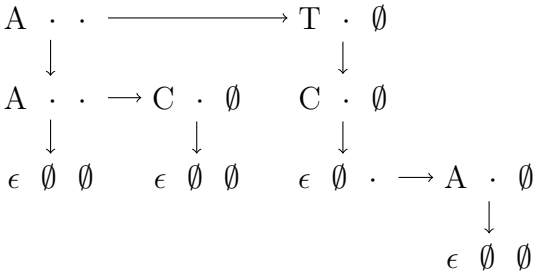
Question 2.2.1 Encoder les 4 mots AA, AC, TCA et TC dans un 4-trie (avec tous les liens, mêmes ceux étant vides).

Question 2.2.2 Dessiner le 4-trie (avec tous les liens, mêmes ceux étant vides) construits sur les mots suivants : TACG ; AAT ; AT ; CGGA et TAC.

On se rend compte que cet arbre contient beaucoup de pointeurs vers le trie vide, et beaucoup de nœuds internes vides (aucun mot ne se termine en ce nœud). L'idée des arbres de la Briandais, pour

représenter les R-tries, consiste à remplacer le tableau de taille R de chaque nœud par une liste **triée** ne contenant que les lettres utiles. On a besoin d'un nouveau caractère (ϵ) pour indiquer la fin d'un mot. On suppose que les frères sont ordonnés selon l'ordre alphabétique : $\epsilon < A < C < G < T$. Le pointeur vers `Nil` est représenté par $\emptyset$.

Ainsi, les mots AA, AC, TCA et TC sont encodés en arbre de la Briandais ainsi :



Question 2.2.3 Encoder l'ensemble des mots de la question 2 dans un arbre de la Briandais.

Question 2.2.4 Donner les spécifications des primitives afin de construire un arbre de la Briandais (par succession d'ajouts de mots), d'y faire la recherche d'un mot et la suppression d'un mot.

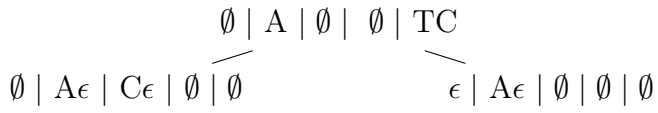
Question 2.2.5 Donner le pseudo-code de l'insertion d'un mot dans un arbre de la Briandais.

Question 2.2.6 Donner le pseudo-code de la suppression d'un mot dans un arbre de la Briandais.

Exercice 2.3 ***: PATRICIA Trie***

Le but des arbres PATRICIA (Practical Algorithm To Retrieve Information Coded In Alphanumeric) est de réduire la taille des R-tries tout en conservant une recherche efficace. Pour ce faire, plutôt que chaque nœud interne permette de distinguer une lettre, il permet de distinguer la plus longue sous-chaîne de lettres commune à plusieurs mots.

Les 4 mots AA, AC, TC et TCA sont représentés par l'arbre PATRICIA suivant :



On remarque que le premier caractère permet toujours de distinguer les sous-chaînes stockées dans chaque nœud interne.

Question 2.3.1 Représenter l'arbre PATRICIA des mots : TACG ; AAT ; AT ; CGGA et TAC. Le comparer à l'arbre de la Briandais de l'exercice précédent.

Question 2.3.2 Donner les spécifications des primitives afin de construire un arbre PATRICIA (par succession d'ajouts de mots), d'y faire la recherche d'un mot et la suppression d'un mot.

Question 2.3.3 Donner le pseudo-code de l'insertion d'un mot dans un arbre PATRICIA.

Question 2.3.4 Donner le pseudo-code de la suppression d'un mot dans un arbre PATRICIA.

Question 2.3.5 Donner le pseudo-code de la fusion de deux arbres PATRICIA en un seul.